计算机科学
COMPUTER SCIENCE

# 基于局部加密网格的LBM程序负载均衡方法研究

黄晨希, 李家辉, 颜辉, 钟英, 卢宇彤

引用本文

黄晨希, 李家辉, 颜辉, 钟英, 卢宇彤. 基于局部加密网格的LBM程序负载均衡方法研究[J]. 计算机科学, 2025, 52(5): 101-108.

HUANG Chenxi, LI Jiahui, YAN Hui, ZHONG Ying, LU Yutong. Investigation on Load Balancing Strategies for Lattice Boltzmann Method with Local Grid Refinement [J]. Computer Science, 2025, 52(5): 101-108.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.241100169

# 基于局部加密网格的 LBM 程序负载均衡方法研究

黄晨希　李家辉　颜　辉　钟　英　卢宇彤
中山大学计算机学院　广州 510006
国家超级计算广州中心　广州 510006
(chenxi.huang@nscc-gz.cn)

**摘　要**　基于局部加密网格的格子玻尔兹曼方法(LBM),在大规模的非稳态计算流体力学问题中有着越来越广泛的使用。局部网格加密方法虽然可以有效减少计算量,但是也带来了严重的负载均衡问题。尤其是在大规模并行计算中,负载均衡算法的选择对整体的计算效率有重要的影响。使用自研的 LBM 程序,简单对比了开源框架 Palabos 的实现结果,然后探讨了基于局部加密网格格子玻尔兹曼方法的静态负载均衡算法。程序中分别基于全局负载和分层负载,使用贪心算法和空间填充曲线算法实现了多种负载均衡策略,提出了计算节点负载均衡优化,并使用小球绕流算例和 DrivAer 算例进行测试和比较。根据计算结果可知,基于分层负载策略的性能比基于全局负载策略的性能要好;同时,贪心算法和空间填充曲线算法对比各有优劣,前者有较好的可扩展性,而后者在进程数较少时效率较高;最后,基于分层负载,结合贪心算法和空间填充曲线算法实现了混合算法,在进程数较多时获得了最佳性能。

**关键词**:格子玻尔兹曼方法;负载均衡算法;局部加密网格;并行计算;节点负载均衡
**中图分类号**　TP319

# Investigation on Load Balancing Strategies for Lattice Boltzmann Method with Local Grid Refinement

HUANG Chenxi,LI Jiahui,YAN Hui,ZHONG Ying and LU Yutong
School of Computer Science and Engineering,Sun Yat-Sen University,Guangzhou 510006,China
National Supercomputing Center in Guangzhou,Guangzhou 510006,China

**Abstract**　The Lattice Boltzmann method(LBM) with local mesh refinement is widely used in large-scale unsteady computational fluid dynamics problems. Although the local mesh refinement method can effectively reduce the computational workload,it also brings serious load balancing issues. Especially in large-scale parallel computing,the choice of load balancing algorithms has a significant impact on the overall computational efficiency. This paper used a self-developed LBM program to compare the implementations of Palabos and to explore the static load balancing algorithm based on the Lattice Boltzmann method with local mesh refinement. This paper implementes various load balancing strategies based on global load and stratified load,uses greedy algorithms and space-filling curve algorithms,and proposes optimization of the load balancing of computing nodes. It uses the flow around a sphere and the DrivAer case as test cases for testing and comparison. Firstly,the results show the performance based on the stratified load strategy is better than that based on the global load strategy. At the same time,a comparative analysis reveals that the greedy algorithm demostrates superior scalability,whereas the space-filling curve algorithm exhibits higher efficiency when operating with a limited number of process. Finally,based on layered load,a hybrid algorithm is implemented by combining greedy algorithm and space filling curve algorithm,which achieves the best performance when the number of processes is larger.

**Keywords**　Lattice Boltzmann method,Load balancing,Local grid refinement,Parallel computing,Node-wise load balancing

## 1　引言

格子玻尔兹曼方法作为一种介观尺度的数值方法[1-2],由于其物理背景清晰、简单易编程、并行扩展性好等特点,在计算流体力学的各个领域中有广泛的应用研究,包括空气动力学、多孔介质、多相流等。标准的格子玻尔兹曼方法通常使用均匀笛卡尔网格进行计算,但是对于具有物理量区域剧烈变化的问题,如空气动力学应用,要捕捉流动剧烈变化区域的流动细节,使用均匀笛卡尔网格会导致网格量大幅增加,严重限制了 LBM 的使用范围。为了解决这个问题,国内外围绕 LBM 的网格技术开展了大量的研究。文献[3]提出了基于树网格的 LBM 算法,大幅提高了计算效率;文献[4]基于非均

到稿日期:2024-11-27　返修日期:2025-02-28
通信作者:李家辉(jiahui.li@nscc-gz.cn)

匀矩形网格结构，提出了25点拉格朗日插值LBM算法，降低了网格数，增加了数值模拟鲁棒性；Marvriplis[5]，Patil等[6]使用多重网格方法，让粗网格遍布整个计算域，通过重叠让粗细网格进行信息交换，从而提高效率。文献[7-9]使用了多区块方法，在粗细网格交界进行数据交换。这些实现中，基于标准LBM的局部网格加密方法主要有多重网格方法和多区块方法。多区块相对于多重网格法，实现更复杂，但内存消耗更少。本文使用多区块的局部网格加密方法，后续关于负载均衡的讨论也将基于此方法展开。

多区块方法把流动区域划分为多块结构化网格，不同的区块根据物理量变化的剧烈程度，使用不同的网格密度。这种方法在保证计算精度的同时，可以有效降低计算量，因此得到了广泛的应用。但是这种方法也带来了一些问题，其中就包括负载均衡问题。一方面，若各个网格块处于不同的层，一个物理时间步内不同层的计算消耗不同，则较难将负载网格块均匀地分配给各个处理单元；另一方面，非均匀网格块之间的拓扑关系更加复杂，难以平衡不同处理单元之间的通信量。在并行计算中，如何把这些计算区块合理分配到不同的计算进程，既要考虑计算负载，也要解决分块带来的通信复杂度的问题。

当下存在许多基于格子玻尔兹曼方法的计算流体力学框架，取得了良好的性能和应用效果，如Palabos[10]，Musubi[11]，waLBerla[12]等。Palabos是基于LBM的通用计算框架，三维局部网格加密使用多区块的算法[9]，基于八叉树数据结构，负载均衡使用基于全局负载的贪心，最大限度保证每个进程都能分到计算任务，但负载分配缺乏局部性；Musubi实现了递归网格加密算法，使用框架TreElM完成负载分配任务，该框架使用空间填充曲线算法进行负载均衡，主要实现了全局的空间填充曲线以及基于各层权重的空间填充曲线算法，但缺乏对分层空间曲线的实现；waLBerla主要面向流体与复杂固体耦合问题，基于分布式八叉树，实现了基于体积单元的局部网格加密算法[13]，对于负载均衡，使用了空间填充曲线算法和调用外部METIS库[14]的方法，提供使用的空间曲线类型包括Hilbert曲线[15]和Z-morton[16]曲线。但上述研究缺少对分层空间填充曲线性能的研究对比。

本文使用自研的LBM并行计算程序，简单对比了开源框架Palabos[10]的实现结果，并针对一些主要的负载均衡算法进行对比，包括贪心算法、空间填充曲线算法[17]；同时在此基础上进行一些优化，并对比了优化后的性能和计算效果。第2章简述构成程序的格子玻尔兹曼方法和并行计算的基础原理和局部网格加密算法；第3章描述负载均衡算法及相应的优化策略；第4章给出了计算结果，首先比较了Palabos框架的实现，然后测试了小球绕流和DrivAer两个三维的算例并加以分析；最后总结全文，并指出进一步的工作方向。

## 2 数值计算方法

本章主要介绍自研LBM并行计算程序使用的主要方法，包括基于Cumulant碰撞模型的格子玻尔兹曼方法、基于八叉树的计算区块划分算法和局部网格加密算法。

### 2.1 格子玻尔兹曼方法

开发的程序主要基于格子玻尔兹曼方法。LBM中宏观物理场的求解主要是由密度分布函数的碰撞和迁移组成的。密度分布函数的控制方程为格子玻尔兹曼方程：

$$f_i(x+\xi_i\Delta t,t+\Delta t)-f_i(x,t)=\Omega_i(f(x,t))+F(x,t) \tag{1}$$

其中，$f_i(x,t)$表示离散速度空间第$i$个分量的密度分布函数，位于离散空间$x$位置，时刻为$t$。式(1)右侧代表局部碰撞算子，左侧表示对流迁移。

开发的程序使用D3Q27模型[18]，速度集表示如下：

$$\frac{\xi_i}{c}=\begin{cases}(0,0,0), & i=0\\ (\pm1,0,0),(0,\pm1,0),(0,0,\pm1), & i=1,\cdots,6\\ (\pm1,\pm1,0),(0,\pm1,\pm1),(\pm1,0,\pm1), & i=7,\cdots,18\\ (\pm1,\pm1,\pm1), & i=19,\cdots,26\end{cases} \tag{2}$$

格子速度使用$c=\Delta x/\Delta t$，宏观密度和动量可以通过分布函数的零阶矩和一阶矩计算。

$$\rho=\sum_i f_i \tag{3}$$

$$\rho\boldsymbol{u}=\sum_i \boldsymbol{\xi}_i f_i \tag{4}$$

其中，$\boldsymbol{u}$代表流体宏观速度，压强通过状态方程计算，即：

$$p=\rho c_s^2 \tag{5}$$

$Cs$代表声速，在D3Q27模型中是常数。

$$c_s=\left(\frac{\mathrm{d}p}{\mathrm{d}\rho}\right)^{1/2}=\frac{c}{\sqrt{3}} \tag{6}$$

平衡态分布函数一般用如下宏观变量表示：

$$f_i^{\mathrm{eq}}(x,t)=w_i\rho\left(1+\frac{u_\alpha\xi_{i\alpha}}{c_s^2}+\frac{u_\alpha u_\beta\xi_{i\alpha}\xi_{i\beta}}{2c_s^4}-\frac{u_\alpha u_\alpha}{2c_s^2}\right) \tag{7}$$

其中，$\alpha$和$\beta$表示各个坐标方向，遵守求和约定，$w_i$表示D3Q27模型下的权重。

$$w_i=\begin{cases}8/27, & i=0\\ 2/27, & i=1,\cdots,6\\ 1/54, & i=7,\cdots,18\\ 1/216, & i=19,\cdots,26\end{cases} \tag{8}$$

为保证高雷诺数下求解的稳定性，$\Omega_i$采用了文献[19]中的cumulant模型，密度分布函数$f$被拉普拉斯变换后的cumulant变量取代。cumulant变量定义在频域空间：

$$F(\Xi)=\mathcal{L}(f(\xi))=\int_{-\infty}^{\infty}f(\xi)\mathrm{e}^{(-\Xi\cdot\xi)}\mathrm{d}\xi \tag{9}$$

$$k_{\alpha\beta\gamma}=c^{-(\alpha+\beta+\gamma)}\frac{\partial^\alpha\partial^\beta\partial^\gamma\ln(F(\Xi,Y,Z))|_{\Xi=Y=Z=0}}{\partial\Xi^\alpha\partial Y^\beta\partial Z^\gamma} \tag{10}$$

每个cumulant阶数为$\alpha+\beta+\gamma$，满足伽利略不变性，cumulant的碰撞过程如式(11)所示：

$$k_{\alpha\beta\gamma}^{*}=k_{\alpha\beta\gamma}-\omega_{\alpha\beta\gamma}(k_{\alpha\beta\gamma}-k_{\alpha\beta\gamma}^{\mathrm{eq}}) \tag{11}$$

其中，$\omega_{\alpha\beta\gamma}$表示松弛速率，$k_{\alpha\beta\gamma}^{\mathrm{eq}}$表示平衡态的cumulant。

运动粘度如下：

$$\upsilon=c_s^2\left(\frac{1}{\omega_1}-\frac{1}{2}\right)\Delta t \tag{12}$$

其中，$\omega_1$表示和物理粘性相关的松弛速率，参考文献[19]，松弛时间$\tau$可表示为：

$$\tau=\frac{1}{\omega_1} \tag{13}$$

## 2.2 八叉树及并行块划分

运行在并行计算机系统时，需要考虑将整个区域的网格按区域位置划分给不同的处理单元，每个处理单元分配得到对应的负载计算。把负载分配给不同的处理单元，避免串行情形下单个处理单元过多的内存需求和占用。本文介绍的程序使用 MPI 并行，是 SPMD 的模式。程序只处理三维问题，并行原理和流程如下。

基于三维空间，根据输入文件和参数定义八叉树划分区域，自顶向下地将需要加密区域对应的位置分割成 2 * 2 * 2 的 8 小块，每个块作为叶子节点；根据配置文件重复该过程，对满足要求的叶子节点往下分，直到区域网格满足层数和精度要求。

以上一步划分的区块 block 为单位，将各个 block 分配给不同 MPI 进程，每个进程记录本地负责处理的 block；每个 block 可以调整格子数量，一般统一设置立方格子大小为 $n*n*n$（不同层的 block 大小统一，只是计算的物理区域范围不同）；格子数量太多容易使得网格填充效率低，过少则会造成通信量过大，本文取 $n=16$，即每个 block 一共有 4 096 个格子。二维示意图如图 1 所示，如进程 $P_0$ 保存固定的 5 个 block 信息并分配相应的内存，计算后同相邻 $P_1$ 和 $P_2$ 的 block 进行数据交换。

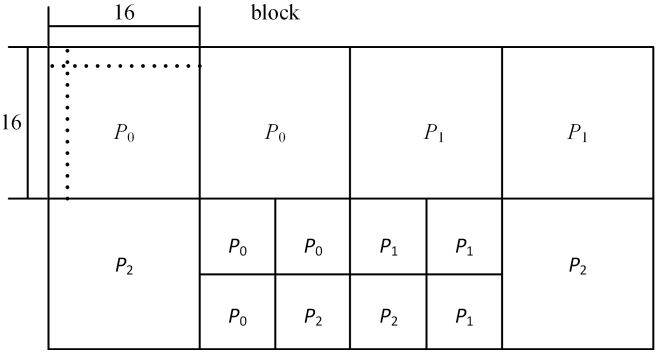



图 1 八叉树网格分配示意图（二维）

Fig. 1 Distribution of grids under octree(2D)

block 之间通过 ghost 层与相邻 block 通信。各个进程均保有任一 block 与 block 之间重叠部分的信息，相当于元数据，需要通信时可以方便地发起和接收。当相邻 block 都属于一个进程时，程序可以将其合并成一个 block，如图 1 中第一层的 $P_0$ 和 $P_1$ 各自拥有的两个 block 可以被合并。合并后去掉额外的 ghost 层，避免多余通信。block 合并后各个进程才分配相应内存。

## 2.3 网格加密算法

网格加密技术要求不同粗细的网格相互通信，同时保证计算精度。不同于均匀网格可以直接交换数据（密度分布函数 $f$），网格在不同尺度的网格间交换数据需要额外处理。一般来说，粗网格到细网格需要插值，添加更多信息到细网格；细网格到粗网格一般是某种形式的平均，过滤掉一些粗网格无法识别的信息。LBM 中多区块局部网格加密技术中基于顶点（Vertex-centered）的方法要求粗细网格重合，通过顶点插值的方法实现；基于体积单元（Cell-centered）的方法通过体积平均保证粗细网格间的密度分布函数守恒。目前主要使用 cell-centered 的方法，开发的程序参考了文献[9]。

网格加密中对粗细网格使用固定的 2 倍关系，即：

$$\Delta x_f=\frac{\Delta x_c}{2} \tag{14}$$

其中，$\Delta x_f$（fine）代表更细一层的网格间距，$\Delta x_c$（coarse）代表更粗一层的网格间距。加密层数增加使得网格间距表示出指数关系。

$$\Delta x_L=\frac{\Delta x_0}{2^L} \tag{15}$$

选择 acoustic scaling，即 $\Delta x_L/\Delta t_L$ 保持常数，对不同层的计算时间步长，有：

$$\Delta t_f=\frac{\Delta t_c}{2} \tag{16}$$

对于交界面上粗网格和细网格重合的点，为保证粘性为常数，有 $\upsilon_c=\upsilon_f$，对不同层的松弛时间 $\tau$ 有：

$$\tau_f=2\left(\tau_c-\frac{1}{2}\right)+\frac{1}{2} \tag{17}$$

对于 post-stream 后的 $f_i^{\mathrm{eq}}(x,t)$，可以分解为：

$$f_i(x,t)=f_i^{\mathrm{eq}}(x,t)+f_i^{\mathrm{neq}}(x,t) \tag{18}$$

其中，$f_i^{\mathrm{eq}}(x,t)$ 由宏观变量计算，对不同粗细的网格，宏观密度和速度是一致的，即 $f_i^{\mathrm{eq}}(x,t)$ 在不同层是一致的，而 $f_i^{\mathrm{neq}}(x,t)$ 正比于 $\nabla u$，对不同层的该变量，结合松弛时间关系式(12)和式(13)可表示为：

$$f_{i,f}^{\mathrm{neq}}(x,t)=\frac{\tau_f}{2\tau_c}f_{i,c}^{\mathrm{neq}}(x,t) \tag{19}$$

结合式(13)，对粗细交界点，当从粗网格到细网格传递 $f$ 时，有：

$$f_{i,f}(x,t)=f_{i,f}^{\mathrm{eq}}(x,t)+\frac{\tau_f}{2\tau_c}f_{i,c}^{\mathrm{neq}}(x,t) \tag{20}$$

其他点使用立方插值，当从细网格传递到粗网格时，有：

$$f_{i,c}(x,t)=f_{i,c}^{\mathrm{eq}}(x,t)+\frac{2\tau_c}{\tau_f}\bar{f}_{i,f}^{\mathrm{neq}}(x,t) \tag{21}$$

整个过程大致使用如图 2 所示的递归运算过程，图中的数据交换略去。

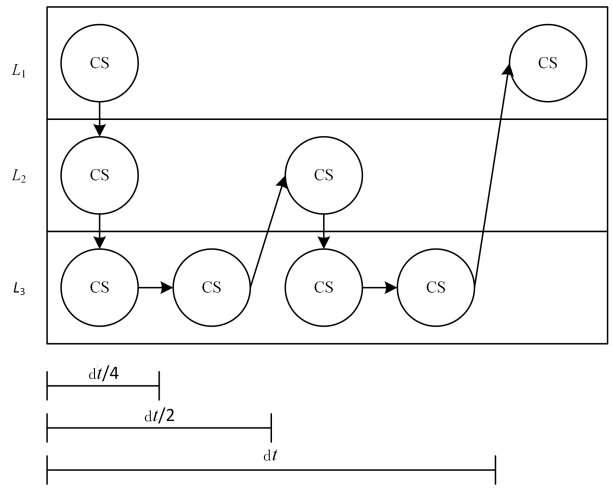



图 2 局部网格加密 LBM 计算流程（3 层）

Fig. 2 Basic process of LBM with grid refinement(3 layers)

## 3 LBM 并行负载均衡算法

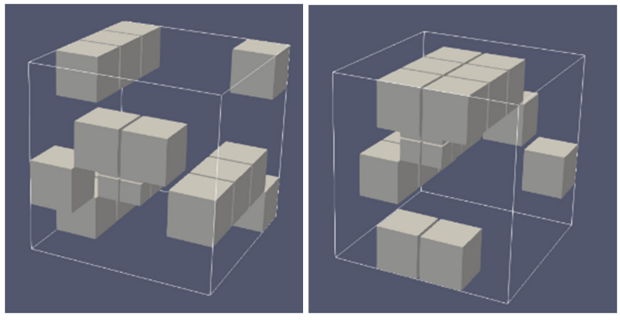
### 3.1 基于全局负载的算法

负载均衡一般包含动态负载均衡算法和静态负载均衡算法。对动态负载均衡算法来说，主要解决两个问题：1)如何分配负载，包含如何分区(partitioning)以及如何将区块负载映射到各个处理单元上(mapping)；2)运行时负载随迭代步数发生变化，需要考虑何时进行负载均衡。本文不考虑计算时动态调整网格的情形。在静态条件下，只需要在初始化前分配好网格和负载，就可以确保运行时计算效率不发生明显变化，因此本文仅讨论静态负载均衡。静态负载均衡算法通过分区(partitioning)和负载分配(mapping)，决定各个进程分到的负载量以及各个进程之间的通信量，进而影响程序性能。下文主要考虑如何分区以及如何将负载分配给各个处理单元(进程)。

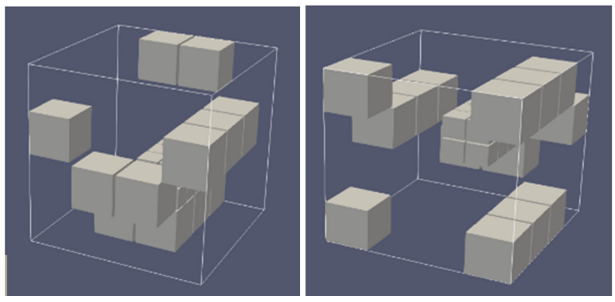
最基本的负载均衡算法是基于进程的全局负载的贪心算法。该算法对所有进程按照已经分配的负载量进行排序，结合 2.1 节描述的 CS 过程和 2.3 节描述的局部网格加密计算流程，在一个物理时间步中，对层数 $L$ 的 block 需要赋值权重 $2^L$；网格最细的层在一次递归运算中处于最深一层，需要迭代次数最多，block 被赋予指数级权重，相当于拥有最多的负载。因此最好从最细层开始遍历八叉树，将网格赋给当前拥有最小负载的进程，然后更新全局进程的负载排序情况，后面以此分配。分配后视情况尝试合并相邻的 block。这种算法可以较好地满足扩展性和可用性，即总是能让拥有最小负载(包括负载为 0)的进程尽快分到计算任务；缺点主要是，各个 block 分到各个进程，没有连续性，不仅会带来较多通信负担，还降低了负载的局部性。对于各个进程所负责的 block，如果空间上是相邻的，则可以将它们合并，但贪心算法很难有效利用这点。由于基于全局负载，该算法对各个进程分层的负载也缺乏控制。

对贪心算法来说不考虑分区，主要考虑如何将负载分配到(mapping)处理单元。其他算法需要考虑分区的问题，分区主要目标是寻找一种算法使各个区块之间的总通信量最小，这是一个 NP 完全问题，只能通过启发式算法来寻找次优解，主要包括 space filling curves(SFC)[17]，recursive coordinate bisection(RCB)[20]，recursive spectral bisection[21]，METIS (multilevel $k$-way)[14]。不同分区效果是类似的，在该类程序和实现[10-12]中广泛使用的是空间填充曲线(space filling curves)算法。本文分区算法主要考虑并使用 Z-Morton 和 Hilbert 的 space filling curve；其他空间填充曲线如 Peano 曲线，效果是类似的。SFC 算法一般将空间中的块沿填充曲线的方向编码，按编码顺序组织到一个一维向量中，然后对这个向量按段分给各个进程，使各个进程分到连续的块，提高局部性并且降低通信量。SFC 比较简单的实现是基于全局的空间曲线，然后按块的数目直接均匀划分，自研程序实现了该算法。

如图 3 和图 4 所示，以 4 进程、两层八叉树分配一个三维空间为例，对比了简单情形下，基于全局负载的贪心算法和空间填充算法的分配效果。由图可知，空间填充曲线相比于贪心算法，负载的局部性和连续性有很大提高。



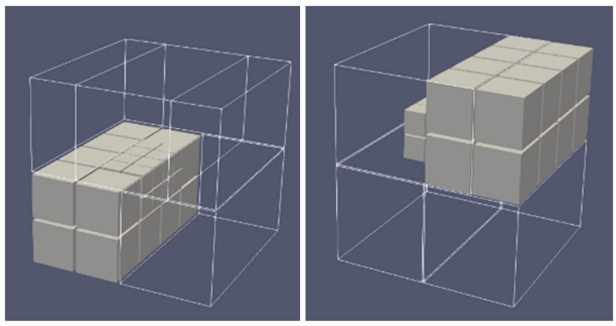
(a)0 号进程分配到的 blocks (b) 1 号进程分配到的 blocks



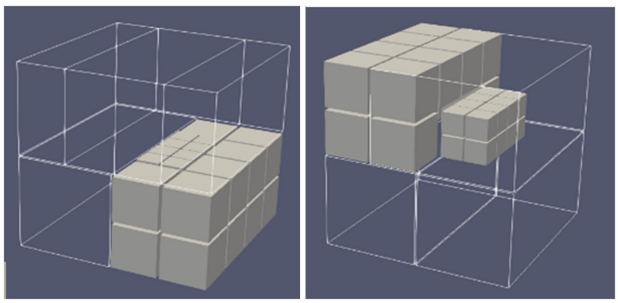
(c)2 号进程分配到的 blocks (d)3 号进程分配到的 blocks

图 3 贪心算法网格分配

Fig. 3 Distribution of grids under greed algorithm



(a)0 号进程分配到的 blocks (b) 1 号进程分配到的 blocks



(c)2 号进程分配到的 blocks (d)3 号进程分配到的 blocks

图 4 空间填充曲线算法网格分配

Fig. 4 Distribution of grids under SFC algorithm

### 3.2 基于分层负载的算法

在使用局部加密网格的 LBM 并行程序中，不同层的计算量和通信量不同。当使用全局负载的贪心算法时，会出现各层之间的负载不均衡；当使用全局的空间填充曲线时，无论按块的数目均分还是按权重均分，更容易出现明显的负载不均衡；因此，需要采用分层负载作为分区和分配指标进行优化。

贪心算法首先可以增加分层负载作为选择指标，当只考虑全局负载均衡时，各层负载容易出现不均衡的情况。在选取负载最小节点时，综合考虑分层负载和全局负载进行选择。进程 $p$ 在网格第 $i$ 层上的负载被定义为一个二元组 $r_{pi}=(l_{pi},g_p)$，$l_{pi}$ 代表该进程在该层的负载，$g_p$ 代表该进程的全局负载。定义该二元组上的小于关系为：给定 $(a_1,b_1)$ 和 $(a_2,b_2)$，当 $a_1<a_2$ 时，$(a_1,b_1)<(a_2,b_2)$；当 $a_1=a_2$ 而 $b_1<b_2$ 时，$(a_1,b_1)<(a_2,b_2)$。即优先比较第一个元素（分层负载）确定顺序，当第一个元素相等时再比较第二个元素（全局负载）确定顺序。算法可以取得分层负载和全局负载的均衡。该算法即分层负载下的贪心算法。

空间填充曲线是全局的空间连续填充曲线，跨越了不同层的网格。当考虑分层时，构建按层填充的空间曲线，由于各层内各个负载单元计算权重相同，因此把每层的块均分给各个进程。这样能保证计算过程中的负载均衡，同时能提高局部性并减小通信。自研程序实现了基于分层负载的 Z-Morton 曲线算法和 Hilbert 曲线算法。

当进程数目较少时，空间填充曲线算法由于通信量小和分配的块局部性强，性能明显高于贪心算法；当进程数增加时，一些网格层内会出现进程分到很少块甚至分不到的情况；随着进程数的进一步增加，整体来看一些进程只能分到一个块。一般除了网格最细的层，其他较粗网格层的块数相对会少一个量级甚至几个量级，许多进程在大部分层都分不到负载。这时空间填充曲线算法带来的低通信量优势会慢慢被消除，其性能趋近贪心算法，而贪心算法由于在负载分配上更平衡，相对表现出更好的性能和扩展。因此，基于分层负载，结合两者的优势进行优化，该混合算法的介绍如下。

首先使用基于分层负载结合计算节点负载均衡的贪心算法分配负载，保证各个进程分到相对平衡的负载量。然后在此基础上，按照分层空间填充曲线定义的顺序，对不同块分到的进程进行有条件的交换，即满足相邻两个进程属于同一计算节点才进行交换，如图 5 所示。通过交换转换为同一进程内部相邻，进一步降低了进程间的通信。

计算节点负载均衡是把具有相邻关系的块分到同一物理节点上，与空间填充曲线类似，主要作用是降低通信量。算法中节点负载均衡和辅助的空间填充曲线块交换可以看作直接空间填充曲线算法的替代。对于相邻的 block，当未被分配到同一进程，但分配到的 MPI 进程属于同一计算节点时，block 之间进行计算节点内部通信，相比于跨节点通信，虽然不能如 2.2 节所述合并一些 block（分到同一 MPI 进程），但也能大大降低通信开销。

实现时赋予每个 block 块 key，为 block 分配进程，如果该 block 的 key 已被分配给某一计算节点，则选取该计算节点，否则将该 key 分配给当前负载最小的计算节点。一个节点的负载是该计算节点中所有 MPI 进程全局负载的总和。对相邻的 block 赋予相同的 key，相邻关系依照如下判定：当 block 拥有相同父节点/祖节点时，同层 block 是相邻的，此时赋予相同的 key。

程序需要定量确定相邻多少 block 为一组赋予相同的 key。给定第 $i$ 层 block 数量 $n_i$ 和计算节点数量 $N$，引入并设计如下经验式：

$$d_i=floor\left(\log_2\left(\frac{n_i}{N}\right)^{\frac{1}{3}}\right) \tag{22}$$

即第 $i$ 层每 $2d_i\times2d_i\times2d_i$ 个 block 应该被赋予同样的 key，其中 $n_i/N$ 是每个节点理论上应负责的第 $i$ 层 block 数量，由于网格是三维的，开 3 次方根得到每一维度的边长，再取 2 的对数得到估计值，最后向下取整。

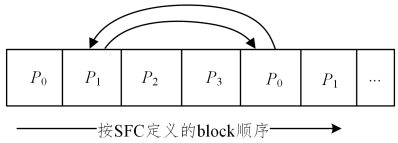



图 5 基于分层负载和节点负载算法示意图

Fig. 5 Schematic diagram of stratified load and node noad-based algorithm

最终这种分配方法经过不同算例测试，能在较少进程数和进程数增加时均取得较优的性能，在不同运行情形下取得较好的平衡。

## 4 计算结果与分析

本章首先基于相同的负载均衡算法对比了自研程序和开源 Palabos；然后在此基础上对程序和算法展开测试，主要测试了两个三维算例，包括简单的三维小球绕流算例和比较复杂的 DrivAer 算例。各个算例先展示了计算结果的可靠性，然后对比各种算法的性能。测试是在天河星逸超算系统上进行的。

### 4.1 Palabos 结果对比

与 Palabos 类似，自研 LBM 并行计算程序主要使用 C++ 开发，并基于 MPI 并行。Palabos 对八叉树网格使用基于全局负载的贪心算法；自研程序在本节使用相同的算法。在核心 CS 部分，自研程序使用了 SoA layout 并通过向量化指令优化，能显著提高性能并更好地适应大规模并行环境。除此之外，使用相同的 gridRefinement3d 算例进行测试，结果如图 6 所示，基于相同的全局贪心负载均衡算法，自研程序性能较 Palabos 平均快 53.6%。后续基于自研程序进一步优化和对比分析。

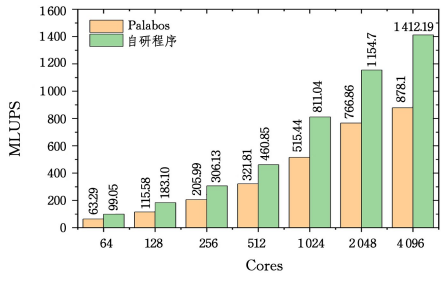



图 6 自研程序和 Palabos 相同负载均衡算法下的性能对比

Fig. 6 Comparisons between self-developed LBM program and Palabos with same load-balancing algorithms

### 4.2 小球绕流算例

对于不同负载均衡算法，首先测试了小球绕流算例，计算

对比小球绕流下的阻力系数 $C_d$，计算参数 $Re=3\,700$，$U=1.0\,\mathrm{m/s}$，$\rho=1.0\,\mathrm{kg/m^3}$。计算使用如图 7 所示的几何文件，小球直径 $d=1\,\mathrm{m}$，小球置于计算空间中央，整个区域大小为 $10\times10\times10$。

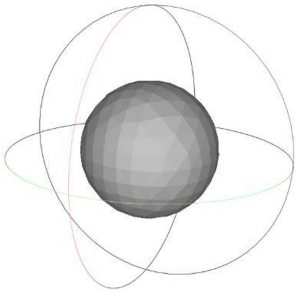


图 7 小球示意图

Fig. 7 Geometry of sphere

网格无关性验证如图 8 所示，$dx$ 达到 9.76 mm 以下后计算精度符合要求。取 $dx=4.88\,\mathrm{mm}$ 的结果，总网格数为 4 000 万，程序计算获得的小球来流方向受力 $F=0.154\,747$，根据式(23)，$A$ 是几何体等效横截面积，后处理获得 $C_d=0.394\,060$。理论值参考文献[22]的理论式，获得 $C_d=0.392\,395$，相对误差仅为 0.42%。计算流场如图 9 所示，符合该雷诺数情况下的流动。

$$c_d=F/\frac{1}{2}\rho U^2 A \tag{23}$$

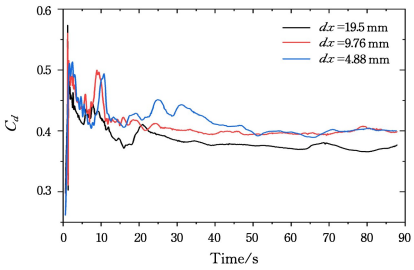



图 8 小球绕流阻力系数随时间的变化

Fig. 8 $C_d$ of sphere varies by time

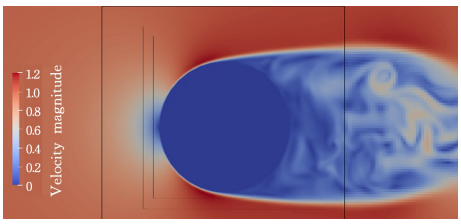



图 9 小球绕流流场速度分布

Fig. 9 Velocity fields of sphere

对于不同算法，使用 $dx=4.88\,\mathrm{mm}$ 约 4 000 万网格，进行强扩展性测试。表 1 和图 10 给出了对几种不同负载均衡算法的比较。不同核数下该算例的运算速度采用单位 MLUPS，含义是每秒更新百万格点数。每个值是取 3 次平均的结果。

从表 1 可以看到，基于全局负载的算法，在进程数较少时，SFC 算法性能略高于贪心算法，速率最多高出 11.3%，但是随着进程数的增加，尤其是当进程数达到 4 096 时，低于贪心算法 14.2%。这是由于当进程数很大时，基于 SFC 算法的每个进程分到很少的 block，效率显著下降。

表 1 各种算法不同核数计算速度(MLUPS)的比较

Table 1 Comparison of MLUPS among different algorithms

| cores | 全局负载贪心 | 全局负载SFC | 分层负载贪心 | 分层负载Morton SFC | 分层负载Hilbert SFC | 分层负载混合算法 |
|---|---|---|---|---|---|---|
| 64 | 99.05 | 97.96 | 100.10 | 136.07 | **138.24** | 136.30 |
| 128 | 183.10 | 183.68 | 189.86 | 242.84 | **250.28** | 208.17 |
| 256 | 306.13 | 321.92 | 321.35 | 378.68 | **383.60** | 363.80 |
| 512 | 460.85 | 512.58 | 486.94 | **591.77** | 581.13 | 586.39 |
| 1 024 | 811.04 | 757.59 | 850.29 | 1 018.86 | **1 031.68** | 1 022.75 |
| 2 048 | 1 154.70 | 1 163.85 | 1 306.90 | 1 379.57 | 1 380.49 | **1 434.73** |
| 4 096 | 1 412.19 | 1 218.79 | 1 539.22 | 1 411.04 | 1 443.09 | **1 563.25** |

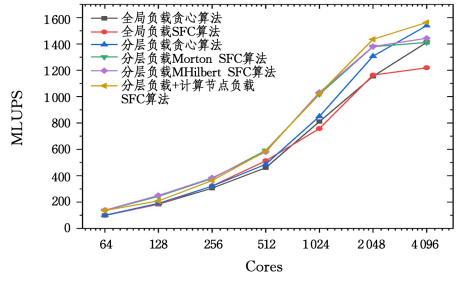



图 10 小球算例下不同负载均衡算法的性能比较

Fig. 10 Comparisons among different algorithms on sphere

而基于分层负载，首先可以看到，相对于全局负载策略，效率有明显提升，对贪心算法效率最多提升 13.2%(2048 核)，平均提高 6.0%；对 SFC 算法，Morton 曲线速率最多提升 38.9%(64 核)，平均提高 24.7%，Hilbert 曲线最多提高 41.1%(64 核)，平均提高 26.2%。然而，与全局负载类似，随着进程数的增加，贪心算法的效率提升要明显大于 SFC 算法，当进程数增加到 4 096 核时，分层负载下贪心算法的效率相对于 Morton SFC 和 Hilbert SFC，分别要快 9.1% 和 6.6%。

考虑到贪心算法和 SFC 算法各有优势，使用了 3.2 节所述的、两者的混合算法，并进行了实现和测试。从表 1 可以看到，进程数较小时，该算法的性能基于 SFC 和贪心算法之间；随着进程数增加，该算法表现出了良好的可扩展性，达到最高的 1 563.25 MLUPS。

### 4.3 DrivAer 算例

接下来使用相对复杂的实际算例 DrivAer 模型。这是一个广泛用于汽车工业领域的算例模型；对比的计算参数 $Re=4.89\times10^6$，入口速度 $U=16\,\mathrm{m/s}$，$\rho=1.225\,\mathrm{kg/m^3}$，使用如图 11 所示的几何模型。

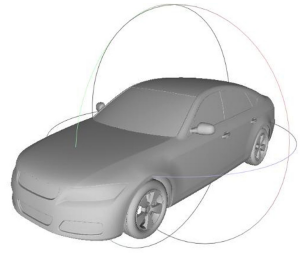


图 11 DrivAer 示意图

Fig. 11 Geometry of DrivAer

网格无关性验证如图 12 所示，可以看到 $dx$ 在 4.88 mm 以下波动范围大大降低，时均值趋于稳定。

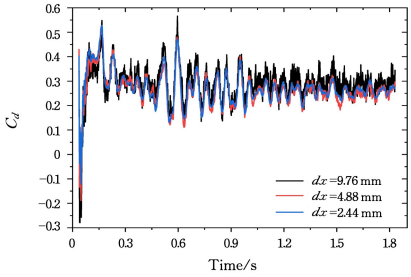



图 12 DrivAer 阻力系数随时间的变化

Fig. 12 $C_d$ of DrivAer varies by time

取 $dx$=2.44 mm 的结果，总网格数 8 亿，根据式(23)，最终获得来流方向阻力系数为 0.2585。同实验值 0.252[23] 对比，相对误差为 2.57%。计算流场如图 13 所示。

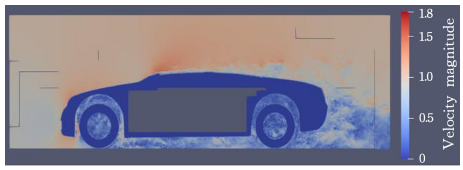



图 13 DrivAer 流场速度分布

Fig. 13 Velocity fields of DrivAer

对不同算法在该算例下开展强扩展性测试，使用 $dx$=2.44 mm约 8 亿的网格。由表 2 和图 14 可知，DrivAer 的结果和小球绕流算例类似，分层负载下算法性能高于全局负载下对应算法，同时贪心算法的可扩展性要优于 SFC 算法，而结合了贪心算法和 SFC 算法的混合算法，在进程数达到 10240 时，获得了最高的效率，超出分层负载贪心算法 5.2%，超出分层负载 Morton 曲线算法 8.8%以及超出 Hilbert 曲线算法 9.9%。

表 2 各种算法不同核数计算速度(MLUPS)的比较

Table 2 Comparison of MLUPS among different algorithms

| cores | 全局负载贪心 | 全局负载 SFC | 分层负载贪心 | 分层负载 Morton SFC | 分层负载 Hilbert SFC | 分层负载混合算法 |
|---|---|---|---|---|---|---|
| 640 | 450.82 | 670.15 | 489.75 | 814.40 | 795.98 | 623.23 |
| 1280 | 811.62 | 1185.97 | 1021.91 | 1286.78 | 1289.21 | 1210.65 |
| 2560 | 1578.40 | 1803.29 | 1619.42 | 2038.61 | 2037.92 | 1939.55 |
| 5120 | 2813.73 | 2367.78 | 2778.28 | 3392.47 | 3247.18 | 3187.39 |
| 10240 | 3528.03 | 2773.07 | 3743.54 | 3618.40 | 3583.65 | 3938.19 |

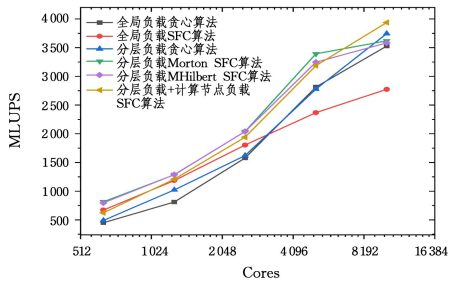



图 14 DrivAer 算例下不同空间填充曲线算法的性能比较

Fig. 14 Comparisons among different SFC algorithms on DrivAer

**结束语** 本文基于自研 LBM 程序，在相同负载均衡算法下对比了 Palabos 开源框架的结果，自研程序在相同情形下相比于 Palabos 性能显著提高；并对比了基于局部网格加密下的静态负载均衡算法，包括基于全局负载的贪心算法和空间填充曲线算法；基于分层负载的贪心算法、Morton 曲线 SFC 算法、Hilbert 曲线 SFC 算法，以及结合了贪心算法和 SFC 算法的混合算法，该算法提出了相邻 block 粒度公式。性能对比主要使用三维小球绕流算例和 DrivAer 算例进行测试。

首先，基于分层负载的算法相对于全局负载的实现，都能显著提高性能；同时，贪心算法和空间填充曲线算法对比各有优劣，前者有较好的可扩展性，而后者在进程数较少时效率较高，这是由于当进程数很大时，空间填充曲线算法的每个进程分到很少的 block，效率显著下降，而贪心算法在进程数较多时，负载较为均衡。

基于以上事实，结合 SFC 算法和贪心算法两者的优势，基于分层负载，使用计算节点负载均衡技术，实现了混合负载均衡算法。相比于分层负载的 SFC 算法，其进程数低时速率略有降低，但是当进程数增加时，性能达到最高，表现出更优的可扩展性。

目前的研究主要基于静态负载均衡开展，下一步将推广到基于动态负载均衡的相关应用中。

## 参 考 文 献

[1] HE Y L, WANG Y, LI Q. Lattice Boltzmann method: Theory and applications [M]. Beijing: Science Press, 2009.

[2] CHEN S, DOOLEN G D. Lattice Boltzmann Method for Fluid Flows[J]. Annual Review of Fluid Mechanics, 1998, 30(1): 329-364.

[3] AN B, SANG W M. The numerical stu-dy of lattice Boltzmann method based on different grid structure[J]. Chinese Journal of Theoretical and Applied Mechanics, 2013, 45(5): 699-706.

[4] AN B, MENG X Y, YANG S J, et al. Research on the lattice Boltzmann algorithm for grid refinement based on non-uniform rectangular grid[J]. Chinese Journal of Theoretical and Applied Mechanics, 2023, 55(10): 2288-2296

[5] MARVRIPLIS D J. Multigrid solution of the steady state lattice Boltzmann equation [J]. Computers & Fluids, 2006, 35: 793-804.

[6] PATIL D V, PREMNATH K N, BANERJEE S. Multigrid lattice Boltzmann method for accelerated solution of elliptic equations[J]. Journal of Computational Physics, 2014, 265: 172-194.

[7] FILIPPOVA O, HÄNEL D. Grid Refinement for Lattice-BGK Models[J]. Journal of Computational Physics, 1998, 147(1): 219-228.

[8] DUPUIS A, CHOPARD B. Theory and applications of an alternative lattice Boltzmann grid refinement algorithm[J]. Physical Review E, 2003, 67(6): 066707.

[9] LAGRAVA D, MALASPINAS O, LATT J, et al. Advances in multi-domain lattice Boltzmann grid refinement[J]. Journal of Computational Physics, 2012, 231(14): 4808-4822.

[10] LATT J, MALASPINAS O, KONTAXAKIS D, et al. Palabos: Parallel Lattice Boltzmann Solver[J]. Computers & Mathema-

tics with Applications, 2021, 81: 334-350.
[11] HASERT M, MASILAMANI K, ZIMNY S, et al. Complex fluid simulations with the parallel tree-based Lattice Boltzmann solver Musubi[J]. Journal of Computational Science, 2014, 5(5): 784-794.
[12] FEICHTINGER C, DONATH S, KÖSTLER H, et al. WaLBerla: HPC software design for computational engineering simulations[J]. Journal of Computational Science, 2011, 2(2): 105-112.
[13] ROHDE M, KANDHAI D, DERKSEN J J, et al. A generic, mass conservative local grid refinement technique for lattice-Boltzmann schemes[J]. Numerical Methods in Fluids, 2006, 51(4): 439-468.
[14] KARYPIS G, KUMAR V. A fast and high quality multilevel scheme for partitioning irregular graphs[J]. SIAM Journal on Scientific Computing, 1998, 20(1): 359-392.
[15] CAMPBELL P M, DEVINE K D, FLAHERTY J E, et al. TERESCO, Dynamic octree load balancing using space-filling curves[R]. Williams College Department of Computer Science Technical Report. 2003.
[16] MORTON G M. A computer oriented geodetic data base and a new technique in the file sequencing[J] International Business Machines Ltd., 1966, 20.
[17] BORRELL R, OYARZUN G, DOSIMONT D, et al. Parallel SFC-based mesh partitioning and load balancing[C]// 2019 IEEE/ACM 10th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems(ScalA). 2019: 72-78.
[18] WHITE A T, CHONG C K. Rotational invariance in the three-dimensional lattice Boltzmann method is dependent on the choice of lattice[J]. Journal of Computational Physics, 2011, 230(16): 6367-6378.
[19] GEIER M, SCHÖNHERR M, PASQUALI A, et al. The cumulant lattice Boltzmann equation in three dimensions: Theory and validation[J]. Computers & Mathematics with Applications, 2015, 70(4): 507-547.
[20] SIMON H D, TENG S H. How Good is Recursive Bisection ?[J]. SIAM Journal of Scientific Computing, 1997, 18(5): 1436-1445.
[21] DRIESSCHE R V, ROOSE D. An improved spectral bisection algorithm and its application to dynamic load balancing [J]. Parallel Computing, 1995, 21(1): 29-48.
[22] CLIFT R, GRACE J R, WEBER M E. Bubbles, drops, and particles[M]. Academic Press, 1978.
[23] HEFT A I, INDINGER T, ADAMS N A. Introduction of a new realistic generic car model for aerodynamic investigations[R]. SAE Technical Paper, 2012.

**HUANG Chenxi**, born in 1994, postgraduate. His main research interests include computational fluid dynamics and high-performance computing.

**LI Jiahui**, born in 1984, Ph.D. His main research interests include computational fluid dynamics and high-performance computing.

(责任编辑:喻藜)